Java™ Persistence API 2.0

Linda DeMichiel, Sun Microsystems

TS-5509

Java

Sun microsystems

Learn about work in progress and new features of the Java™ Persistence API

GOAL

Agenda

- Introduction and Goals
- O/R Mapping and Domain Modeling
- Runtime APIs
- Queries and Other Features in the Queue
- Summary and Roadmap

Java Persistence API 2.0

- Java Specification Request (JSR) 317
  - Launched Fall 2007
  - 22 member Expert Group
- Purpose: to solidify the standard
  - Address requests from the community for needed features
  - Standardize optional functionality, reduce non-portability
  - Add better support for Java EE Platform environments

JSR Proposed Functionality

- More flexible modeling capabilities
- Expanded object/relational mapping functionality
- Additions to Java Persistence query language
- Criteria query API
- Standardization of configuration hints
- Additional contracts for handling detached entities
- Additional contracts for extended persistence contexts
- Support for validation

Agenda

- **Introduction and Goals**
- **O/R Mapping and Domain Modeling**
- Runtime APIs
- Queries and Other Features in the Queue
- Summary and Roadmap

More Flexible Modeling and Mapping

- Collections of basic types
- Improved support for embeddable classes
- Ordered lists
- Generalized maps
- Expanded relationship mapping options
- More flexible use of access types
- . . .

Collections of Basic Types

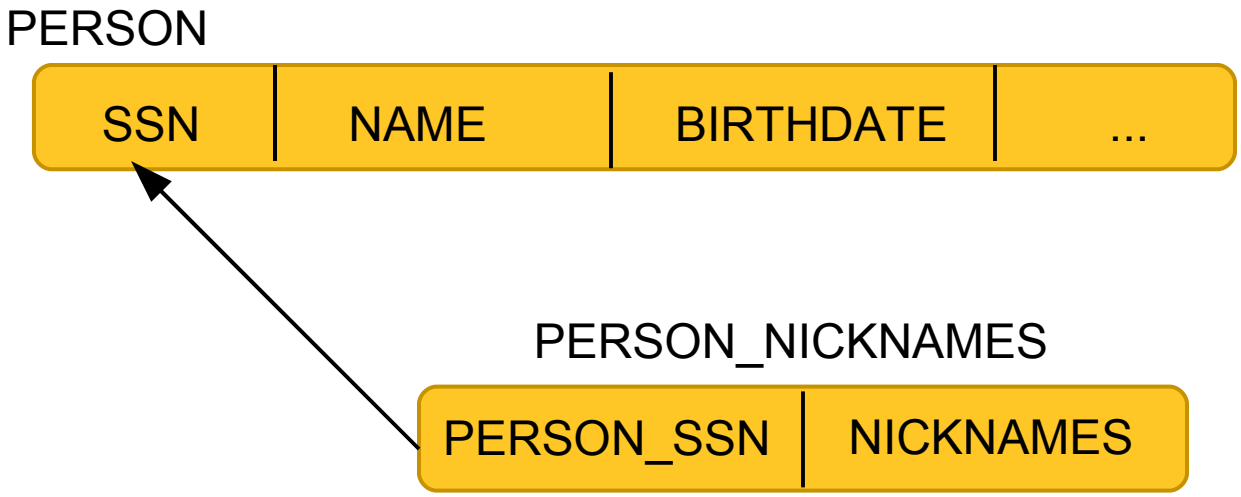
- Collections of strings, integers, etc.

- Specified by @ElementCollection
- Stored in “collection table”
- Customize mapping with
  - @CollectionTable
  - @Column

Collections of Basic Types

```
@Entity
public class Person {
    @Id protected String ssn;
    protected String name;
    protected Date birthDate;
    ...
    @ElementCollection
    protected Set<String> nickNames;
    ...
}
```

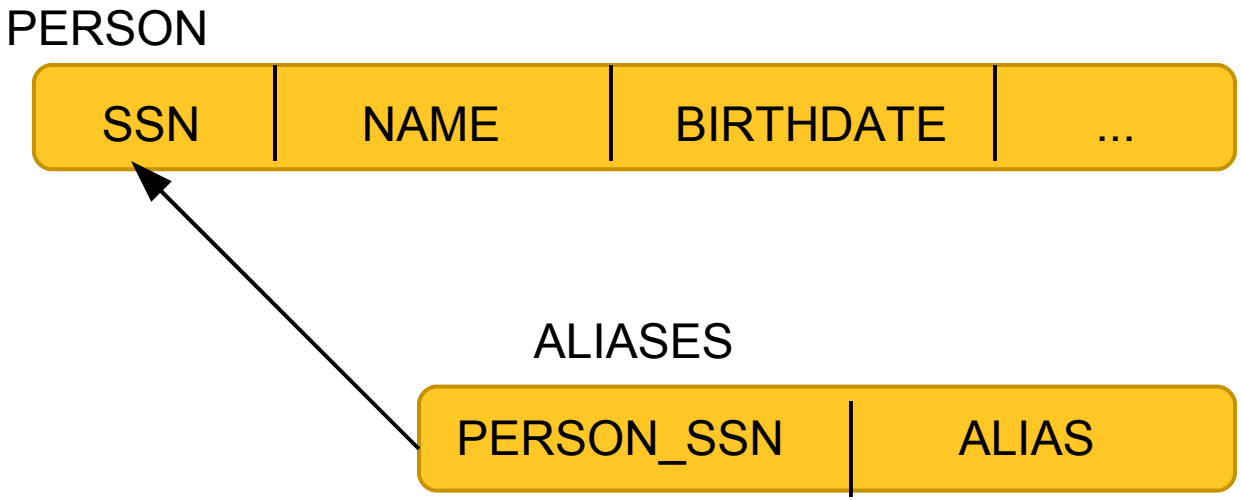
Relational Mapping (default mapping)



Collections of Basic Types

```
@Entity
public class Person {
    @Id protected String ssn;
    protected String name;
    protected Date birthDate;
    ...
    @ElementCollection
    @CollectionTable(name="ALIASES")
    @Column(name="ALIAS")
    protected Set<String> nickNames;
    ...
}
```

Relational Mapping (customized mapping)



Collections of Embeddable Types

- E.g., Address, Detail

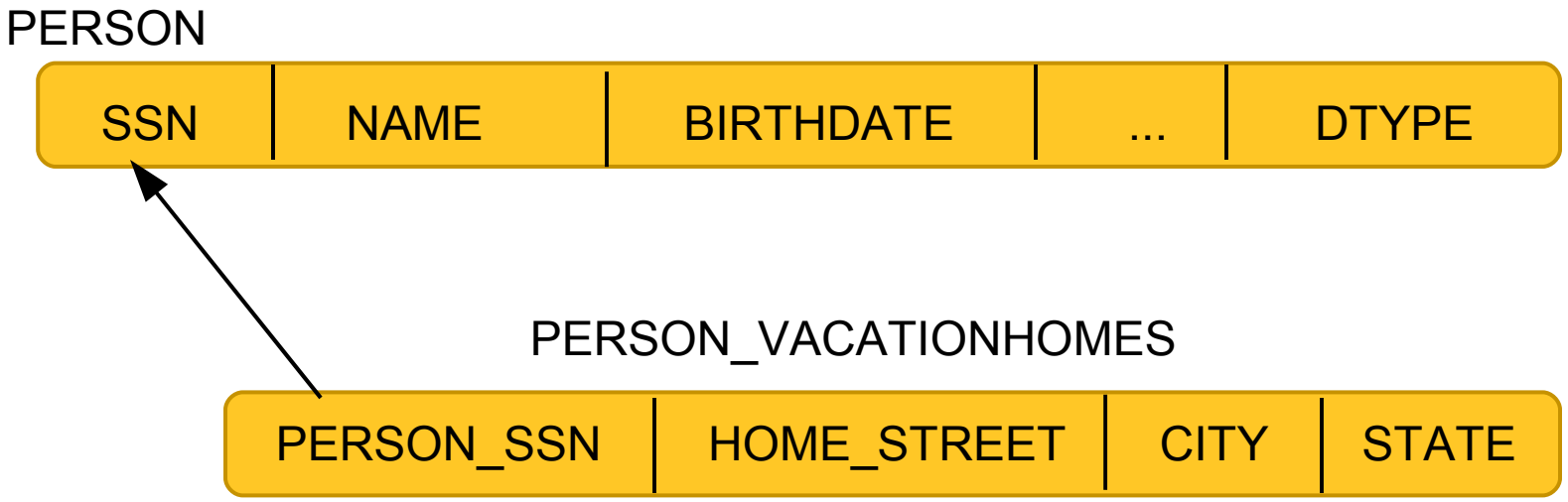
- Specified by @ElementCollection
- Stored in collection table
- Customize mapping with
  - @CollectionTable
  - @AttributeOverride

Collections of Embeddable Types

```
@Entity public class RichGuy extends Person {
    ...
    @ElementCollection
    @AttributeOverride(name="street",
         column=@Column(name="HOME_STREET"))
    protected Set<Address> vacationHomes;
    ...
}

@Embeddable public class Address {
    protected String street;
    protected String city;
    protected String state;
    ...
}
```

Relational Mapping



Multiple Levels of Embedding

- Embeddables can have embeddables as state
- Customize mapping with
  - @AttributeOverride, using dot(“.”) syntax

Multiple Levels of Embeddables

```
@Embeddable public class Address {
   protected String street;
   protected String city;
   protected String state;
   protected ZipCode zipCode;
}

@Embeddable public class ZipCode {
   protected String zip;
   protected String plusFour;
}

@Entity public class Customer {
   @Id protected Integer id;
   protected String name;
   ...
   @AttributeOverride(name="zipCode.plusFour",
       column=@Column(name="BLOCK"))
   protected Address address;
   ...
}
```

Relational Mapping

CUSTOMER

Embeddables and Relationships

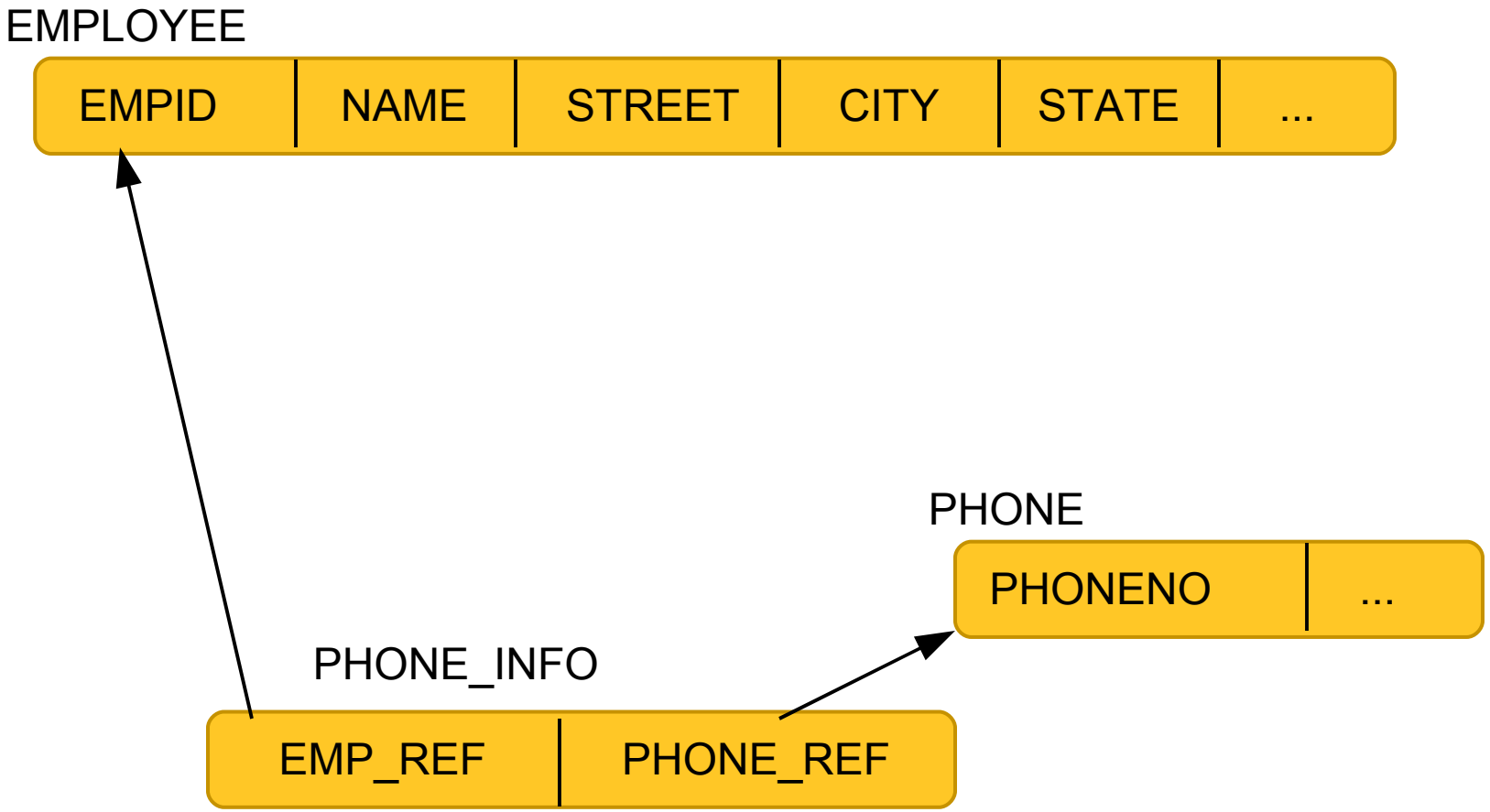
- Embeddables can have relationships
  - @ManyToOne, @OneToOne, @OneToMany, @ManyToMany
- Relationships can be bidirectional
  - Inverse refers to entity containing the embeddable
  - In collections, embeddable must own foreign key side
- Customize relationship mapping with
  - @AssociationOverride

Embeddables and Relationships

```
@Entity public class Employee {
    @Id protected int empId;
    protected String name;
    ...
    @AssociationOverride(
        name="phones",
        joinTable=@JoinTable(
            name="PHONE_INFO",
            joinColumns=@JoinColumn(name="EMP_REF"),
            inverseJoinColumns=@JoinColumn(name="PHONE_REF"))
    )
    protected ContactInfo info;
}

@Embeddable public class ContactInfo {
    Address address;
    @OneToMany Set<Phone> phones;
    ...
}
```

Relational Mapping



Ordered Lists

@OrderBy

- Sort order applied when list is retrieved
- Order isn't “persistent”

@OrderBy

```
@Entity
public class Course {
    ...
    @ManyToMany
    @OrderBy("lastName")
    List<Student> studentsEnrolled;
    ...
}
```

Ordered Lists

@OrderColumn

- Order maintained by additional (integral) ordering column
  - Where column lives is determined by mapping type
- Order is “persistent”

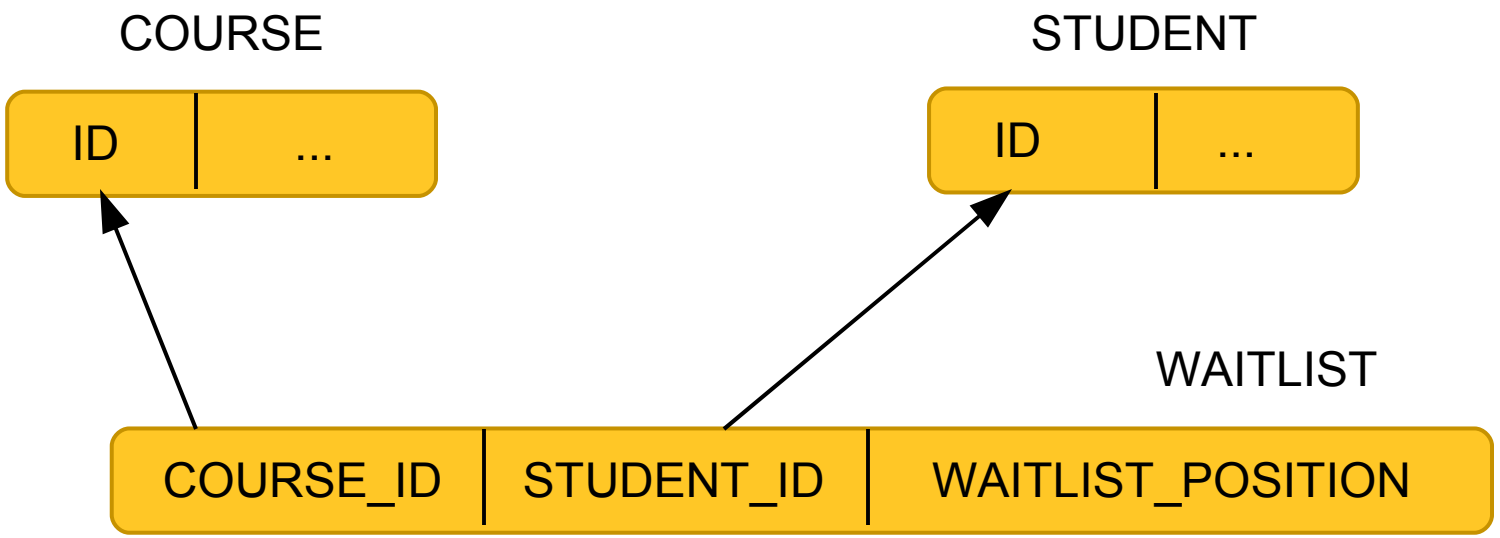
@OrderColumn

```
@Entity
public class CreditCard {
    @Id Long cardNumber;
    ...
    @OneToMany
    @OrderColumn
    List<CardTransaction> transactionHistory;
    ...
}
```

@OrderColumn

```
@Entity
public class Course {
    @Id Integer id;
    ...
    @ManyToMany
    @OrderColumn(name="WAITLIST_POSITION")
    @JoinTable(
        name="WAITLIST"
        joinColumns=@JoinColumn(name="COURSE_ID"),
        inverseJoinColumns=@JoinColumn(name="STUDENT_ID")
    )
    List<Student> waitList;
    ...
}
```

Relational Mapping



Generalized Maps

- Map key can be:
  - Basic type
  - Embeddable
  - Entity
- Map value can be:
  - Basic type
  - Embeddable
  - Entity

Generalized Maps

- Specify Map with
  - @ElementCollection, @OneToMany, @ManyToMany
- Customize Map with
  - @CollectionTable, @JoinTable
  - @Column
  - @AttributeOverride (using “key.” or “value.” syntax)
  - @MapKeyColumn
  - @MapKeyJoinColumn, @MapKeyJoinColumns

Maps

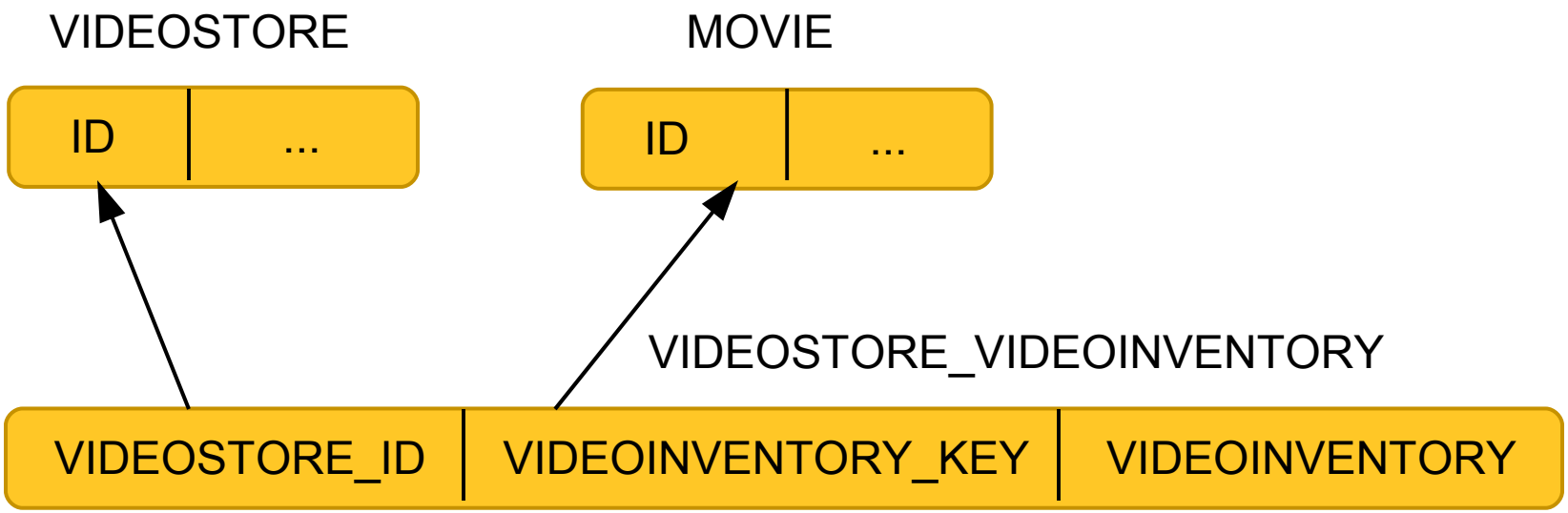
```
@Entity
public class Item {
    @Id int id;
    ...
    @ElementCollection
    // map from picture name to file name
    Map<String, String> photos;
    ...
}
```

Maps

```
@Entity
public class VideoStore {
    @Id int id;
    String name;
    Address location;
    ...
    @ElementCollection
    // map from movie to number in stock
    Map<Movie, Integer> videoInventory;
    ...
}

@Entity
public class Movie {
    @Id int id;
    ...
}
```

Relational Mapping



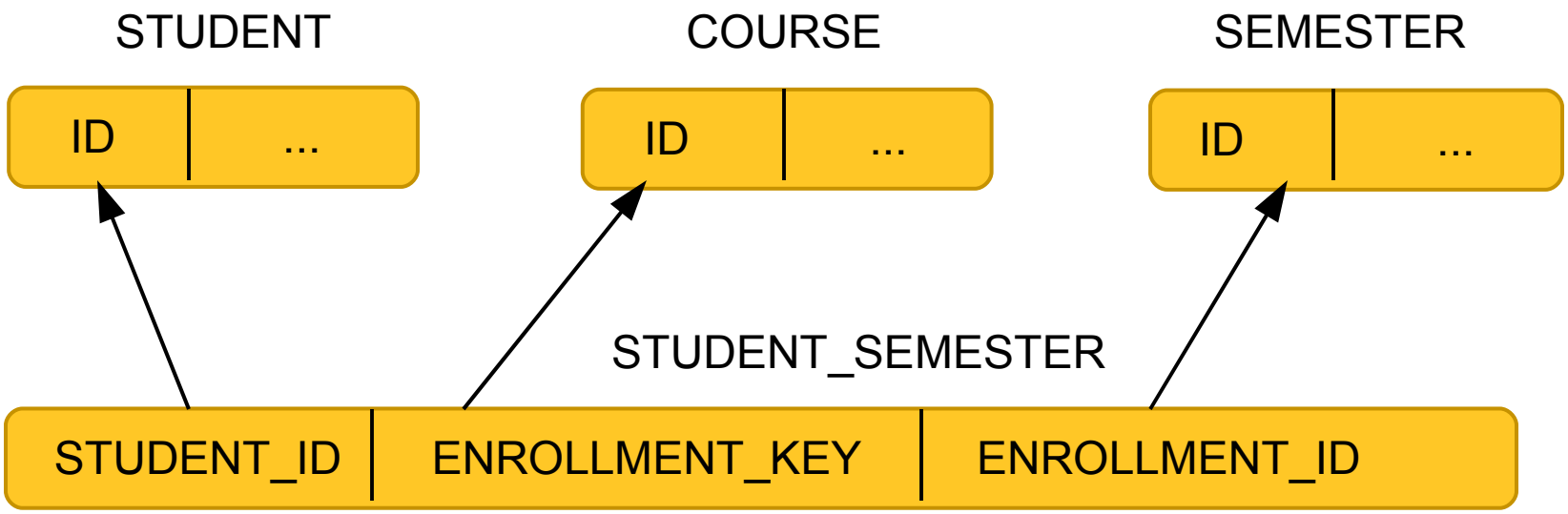
Maps

```
@Entity public class Student {
    @Id int id;
    String name;
    ...
    @ManyToMany   // ternary relationship
    Map<Course, Semester> enrollment;
    ...
}

@Entity public class Course {
   @Id int id;
   ...
}

@Entity public class Semester {
    @Id int id;
    ...
}
```

Relational Mapping



Expanded Relationship Mappings

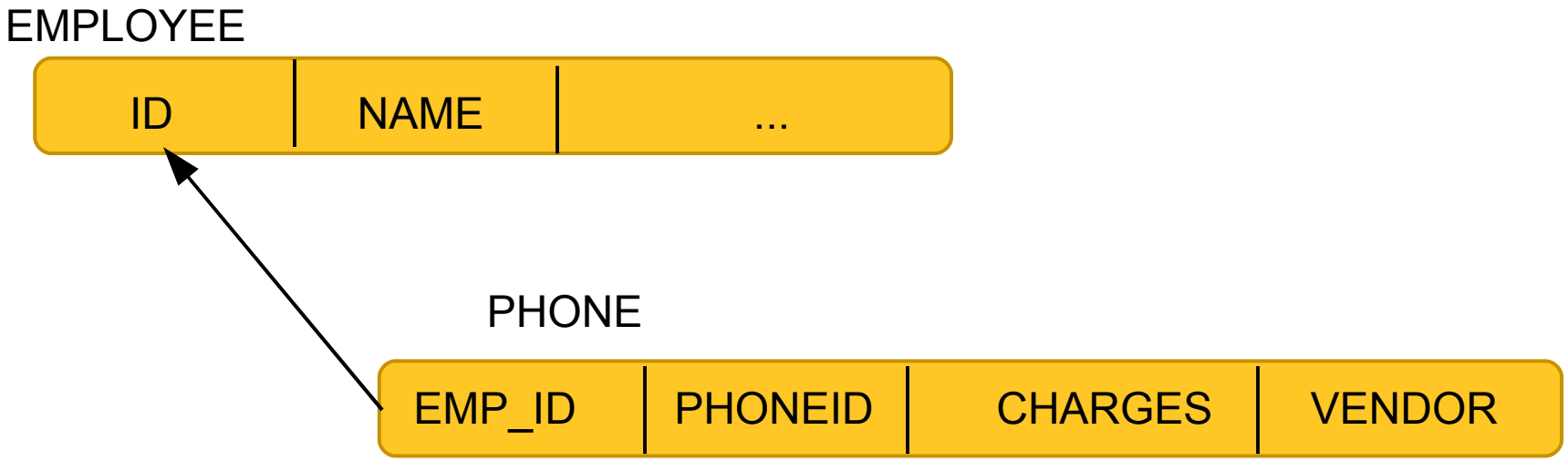
- Foreign key mappings for unidirectional one-to-many relationships
- Join table mappings for
  - One-to-one
  - Many-to-one

Unidirectional One-to-Many Foreign Key Mapping

```
@Entity
public class Employee {
    @Id Integer id;
    String name;
    ...
    @OneToMany
    @JoinColumn(name="EMP_ID")
    Set<Phone> phones;
}

@Entity
public class Phone {
    @Id int phoneId;
    float charges;
    String vendor;
    ...
}
```

Relational Mapping



Inheritance Mapping

- Table per class inheritance mapping optional in Java Persistence API 1.0
- It will remain so

- Unless you tell us otherwise….

Automatic Orphan Deletion

- Deletion of related entities when removed from relationship
- For related entities logically “owned” by “parent”
- For one-to-one and one-to-many relationships

- Specified with orphanRemoval element

- Note: cascade=REMOVE is redundant

Orphan Deletion

```
@Entity
public class Customer {
    @Id int custId;
    Address address;
    ...
    @OneToMany(cascade=PERSIST, orphanRemoval=true)
    Set<Order> orders;
    ...
}

@Entity
public class Order {
    @Id int orderId;
    ...
    @OneToMany(cascade=PERSIST, orphanRemoval=true)
    Set<Item> items;
}
```

Access Type

- Classes in hierarchy can have different access types
- Placement of mapping annotations determines default access for hierarchy
- @Access annotation specifies non-default access
- Access types can be mixed within a class

@Access

```
@Entity
@Access(FIELD)
public class Customer {
    @Id protected int custId;
    protected String name;
    protected Address address;
    ...
    @Transient protected Integer rating;
    ...
    @Access(PROPERTY)
    public Integer getCreditRating() {
      return rating;
    }
    public void setCreditRating(Integer rating) {
      this.rating = rating;
    }
    ...
}
```

Agenda

- **Introduction and Goals**
- O/R Mapping and Domain Modeling
- Runtime APIs
- Queries and Other Features in the Queue
- Summary and Roadmap

Concurrency Control and Locking

- Java Persistence API 1.0 based on optimistic concurrency
  - Database assumed at “read committed” isolation
- Version attributes
  - Prevent conflicting updates
- LockModeType READ
  - Gives repeatable read for locked entity
- LockModeType WRITE
  - Gives repeatable read for locked entity
  - Forces version update

But: optimistic concurrency control may not be good enough!

Pessimistic Locking

- Grab database locks early
- Provider must translate into database locking
  - SELECT FOR UPDATE...
  - Eager writes
- Version attributes not required
  - But important!

- What gets locked ?
  - Entity ?
  - Relationships ?
  - Related objects ?

Lock Modes

- OPTIMISTIC (READ)
- OPTIMISTIC_FORCE_INCREMENT (WRITE)
- PESSIMISTIC
- PESSIMISTIC_FORCE_INCREMENT

Names still undergoing discussion

Locking APIs

- EntityManager methods
  - lock
  - find
  - refresh
- Query methods
  - setLockMode
  - setHint
- @NamedQuery
  - lockMode element

javax.persistence.lock.timeout hint

Pessimistic Locking Failures

- Deadlock
  - Transaction rollback
    - PessimisticLockException
- Lock timeout
  - Statement rollback
    - LockTimeoutException
  - Transaction rollback
    - PessimisticLockException

Other EntityManager and Query Additions

- EntityManager methods
  - clear(entity)
  - getLockMode(entity)
  - getProperties(), getSupportedProperties()
  - getEntityManagerFactory()
- Query methods
  - getMaxResults()
  - getFirstResult()
  - getHints(), getSupportedHints()
  - getNamedParameters(), getPositionalParameters()
  - getFlushMode()
  - getLockMode()

Caching

- Second-level cache
  - Cache for persistence unit
  - Sits between persistence context and database
- Behaviors and APIs are vendor-specific
  - Whether cache enabled or not by default
  - What is cached
  - Caching policies
  - Cache control

Cache API

- Basic functionality over second-level cache
- Methods
  - contains(entityClass, primaryKey)
  - evict(entityClass, primaryKey)
  - evict(entityClass)
  - evictAll()
- Cache mode hints under discussion

Standardized Properties and Hints

So far:

- javax.persistence.lock.timeout
- javax.persistence.query.timeout

- javax.persistence.jdbc.driver
- javax.persistence.jdbc.url
- javax.persistence.jdbc.user
- javax.persistence.jdbc.password

More likely to come....

Agenda

- **Introduction and Goals**
- O/R Mapping and Domain Modeling
- Runtime APIs
- Queries and Other Features in the Queue
- Summary and Roadmap

Java Persistence Query Language

In the queue:

- Functions in SELECT list
- Support for new modeling enhancements
- Restricted polymorphism ?

Criteria API

Goal

- Examine existing APIs
  - Hibernate
  - OJB
  - Cayenne
  - TopLink
  - ...
- Learn from them
- Come up with something “better”

Managing Detachment

- EntityManager.load(entity)
- EntityManager.isLoaded(entity)

- Additional controls for loading entities via
  - Fetch Plans ?
  - Criteria API ?

Validation

- JSR-303 (“Bean Validation”)
  - Goal: define metadata model and API
  - For use in both Java SE platform and Java EE platform
  - Early Draft now available
- Would like to utilize for Java Persistence API

Agenda

- **Introduction and Goals**
- O/R Mapping and Domain Modeling
- Runtime APIs
- Queries and Other Features in the Queue
- Summary and Roadmap

Summary

Java Persistence API 2.0 addresses functionality to support:

- More flexible modeling
- Expanded O/R mapping functionality
- Query language enhancements
- Great portability across implementations
- Alignment with emerging JSRs

Status and Roadmap

- Early Draft Review focused on
  - O/R mapping
  - Modeling
  - Runtime APIs
- Download Early Draft from:
  - http://jcp.org/en/jsr/detail?id=317
- Send Early Draft feedback to:
  - jsr-317-edr-feedback@sun.com
- Next Draft will focus on query area

- Reference Implementation: EclipseLink
- Available with GlassFish project V3

THANK YOU

Linda DeMichiel, Sun Microsystems

TS-5509